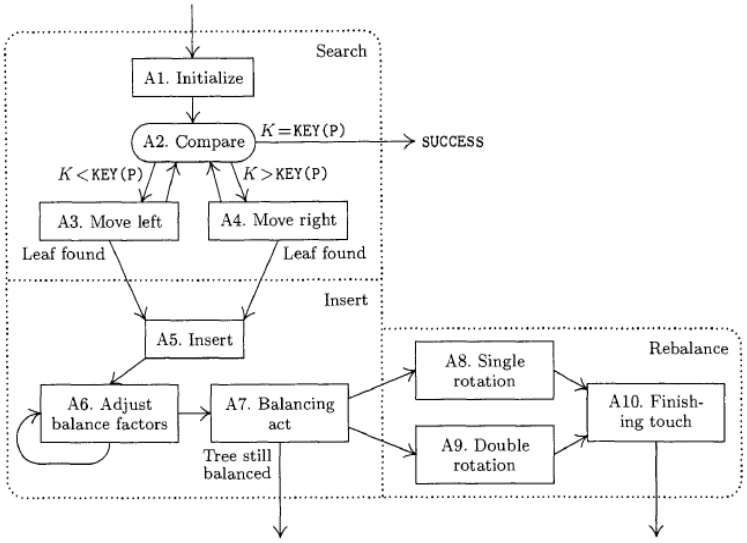
DS LAB- ASSIGNMENT 2 – AVL TREE

Roll No: 214101011

Functions Used:

1. **Insert() :** For Insert, I have applied the "insertion using balance factor" algorithm from the book "The art of programming"

Its flow diagram is given below :



In the program I have marked the various parts using notation A1 to A10.

These points are briefly explained:

A1 : Initialize all the pointer. I have explained the purpose of each ptr in comments.

A2: Compare the key value given and move down the tree.

A3 : If key is less than node's bp , we move left subtree.

A4: If key is greater than node's bp , we move right subtree.

A5: create a new node with key k in the position to insert.

A6: Adjusting balance factor. After inserting the new node, we adjust the BF of all nodes from start to new added.

A7 : Balancing the tree. Here the node bp which is balance point node is used to check if any balancing is needed or not.

A8: Single rotation is done here if slant of bp and its child is same, I.e., both tilted to same side.

A9: Double rotation is done if bp and its child is tilted differently, I.e., B.F of bp is 1 and B.F of ch_bp is –1 or vice versa.

A10: Now that t is new root of subtree , we previously stored parent of bp whose child will now be t.

Note: Check the images for insertion attached at end , and created in folder while running the program .

---

2. Delete() : I have used a lot of idea from insert algorithm and some other source for this method.

Here is high level overview of what I did:-

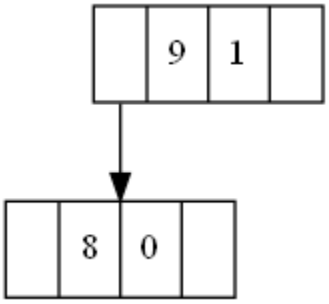
- Initialized all nodes with proper names
- Used a stack to store all visited nodes while traversing to delete a node.
- Find the location of node to delete and push all visited in stack “nodes”.
- If the node to delete has single child , copy contents of that child to parent and delete the child.
- If the node to delete has no child just delete the node.
- If the node to delete has 2 child , first we need to find the successor of node to delete and replace the node to delete with the successor node and delete the successor node.
- After deletion is done, key step is to visit back nodes one by one and set their balance factor, check for imbalance, and fix that imbalance.
- Setting new root(t) as child of par_bp .

The rotation process is almost same as in insertion with slight change in value of direction as here we are deleting instead of inserting , so conditions involving direction are reversed. Else all is same.

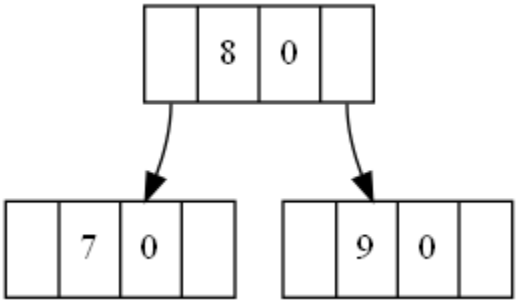
3. Search() - This is straightforward search by key comparison.
4. Print() - This creates a dot file with name given in parameter and creates a gv file which is changed to png fie of same name.
5. deleteTree() - This helper function is called by the destructor when we delete the tree . It deletes all nodes recursively and prints the node its deleting in the output.

Images Generated of tree for Insert in order from i1 to i8 :

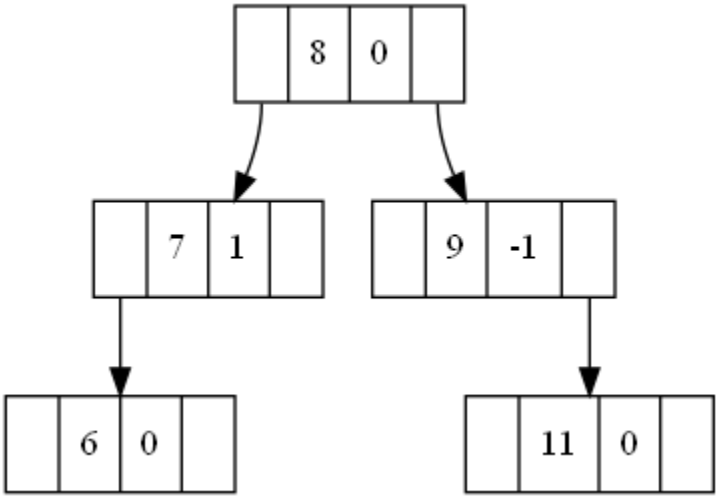
Inserting 2 nodes- i1



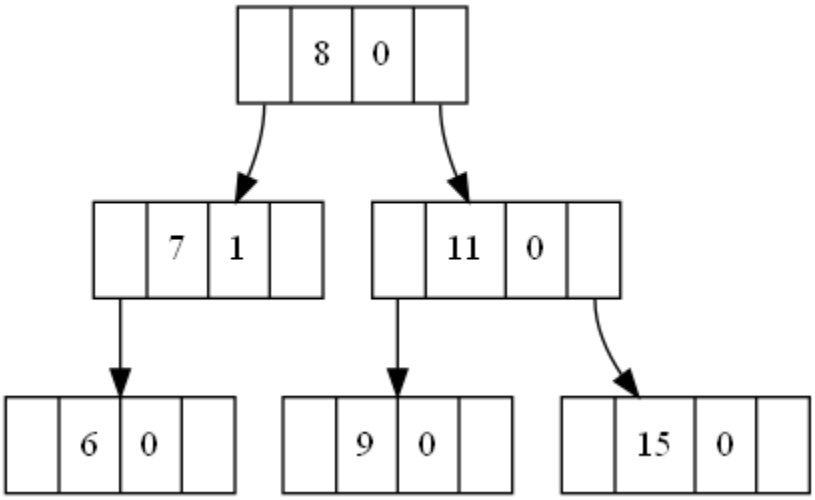
Inserting 7 to get LL imbalance at node 9 - i2



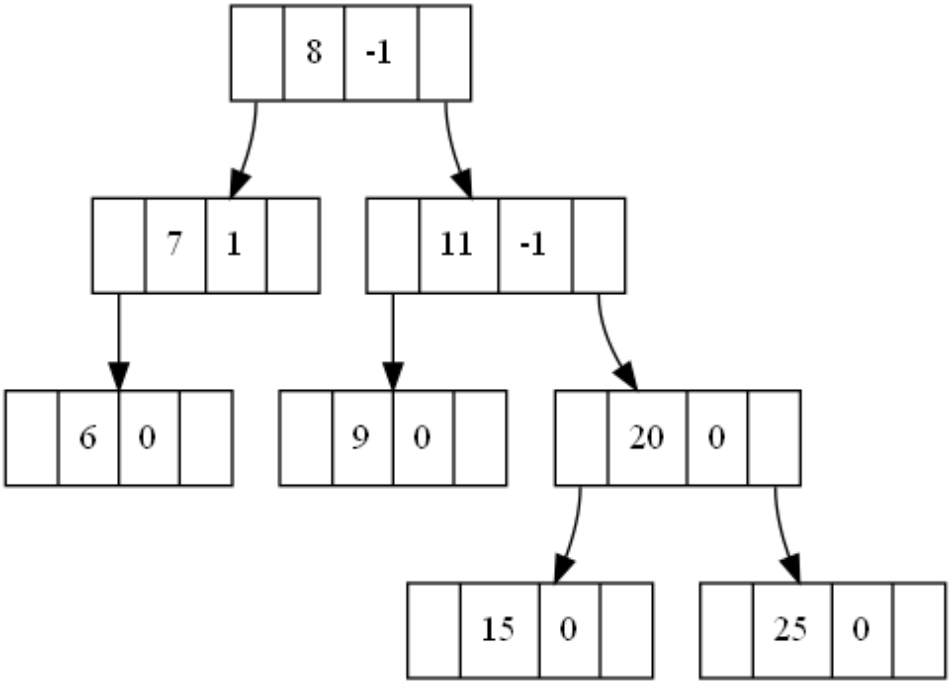
Inserting some more - i3



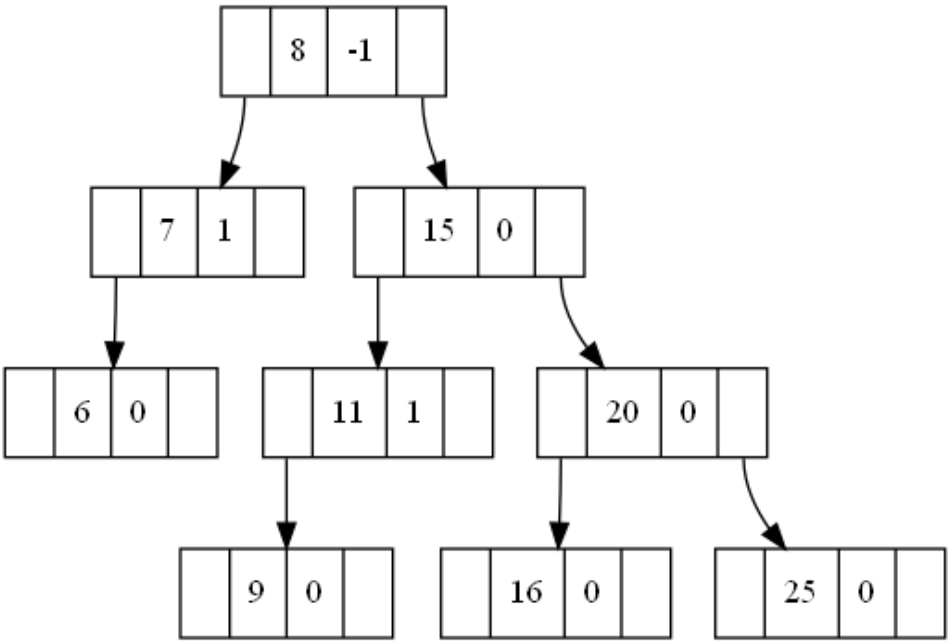
I4 – Inserting 15 to get RR imbalance at node 9



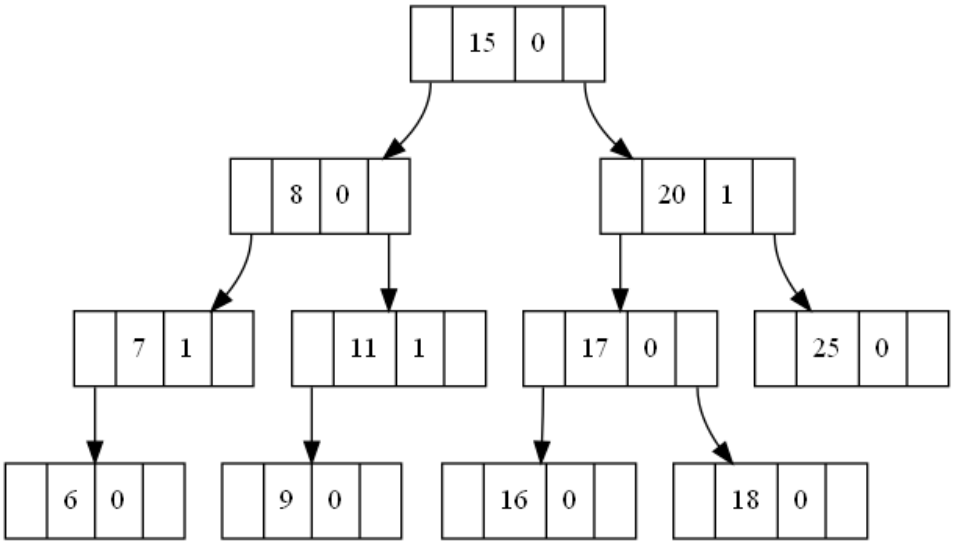
I5 – inserting 20 , 25 , RR imbalance again fixed here at node 15 :



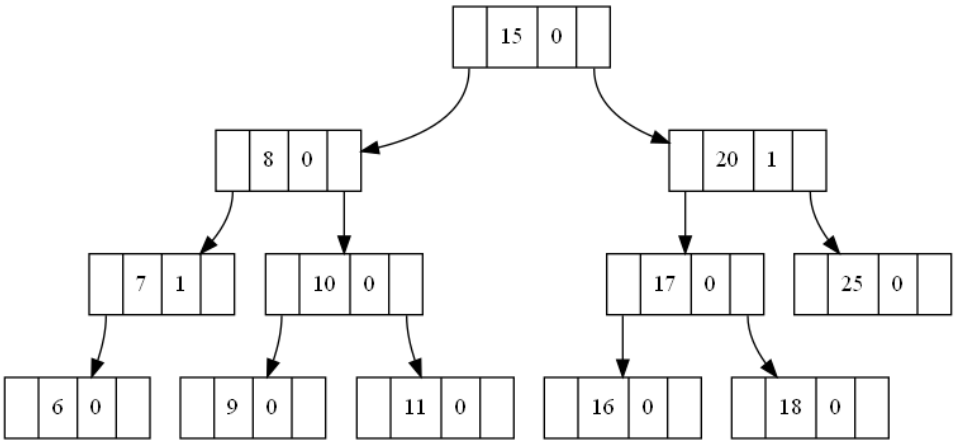
I6 – Inserting 16 , tree gets Rl imbalanced at node 11:



I7 – inserting 17 and 18 creates RR at node 16 then RL imbalance at node 8 :

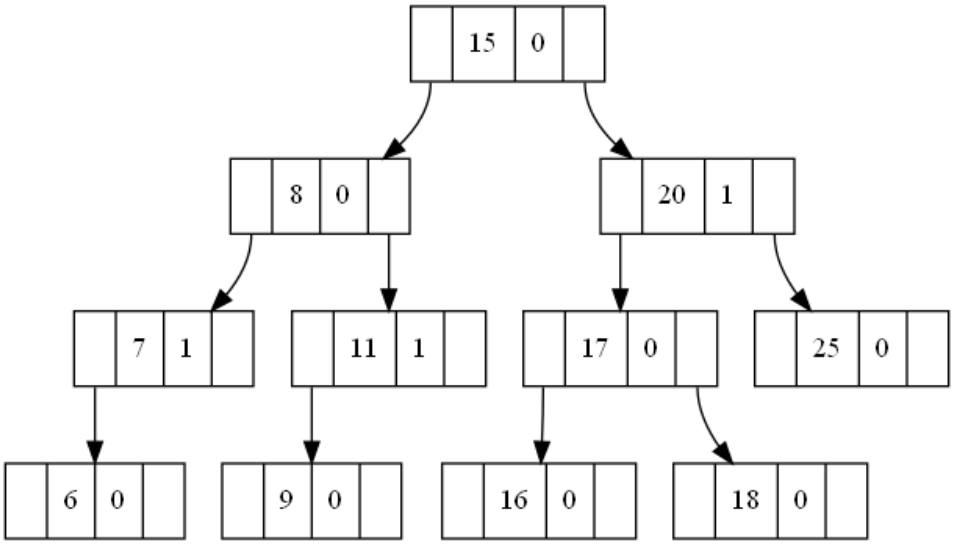


I8 – Inserting 10 created a LR imbalance at node 11.

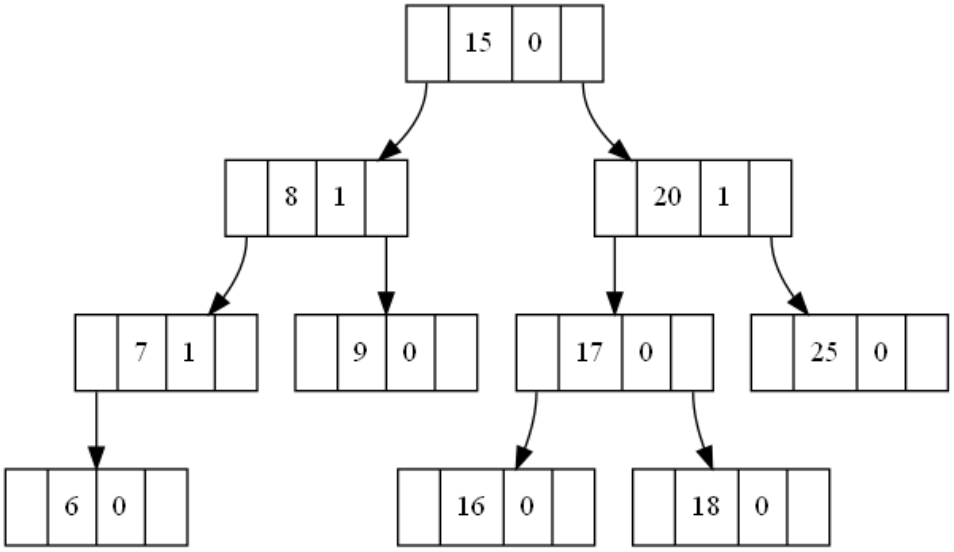


Images of tree d1 to d9 for generated for delete operations :

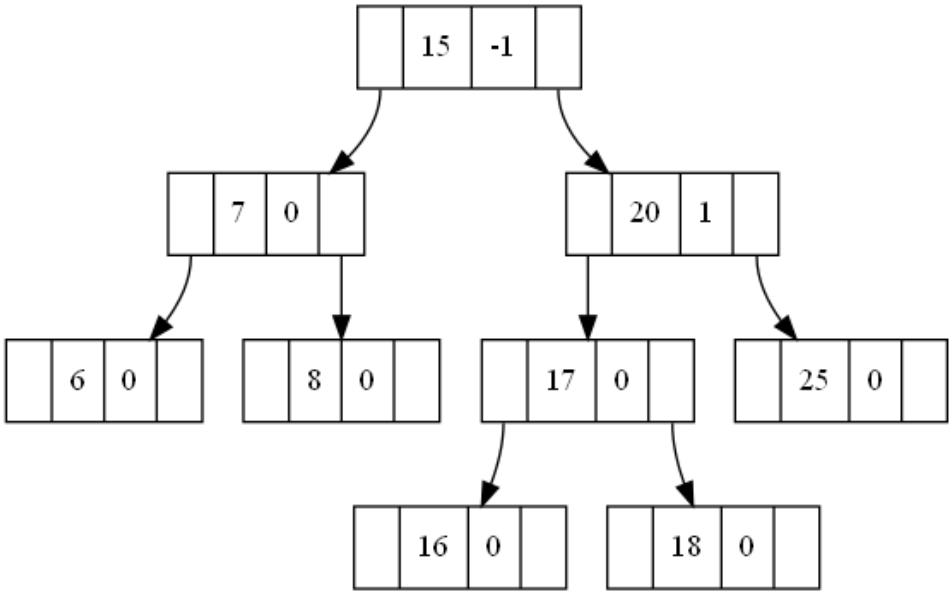
D1



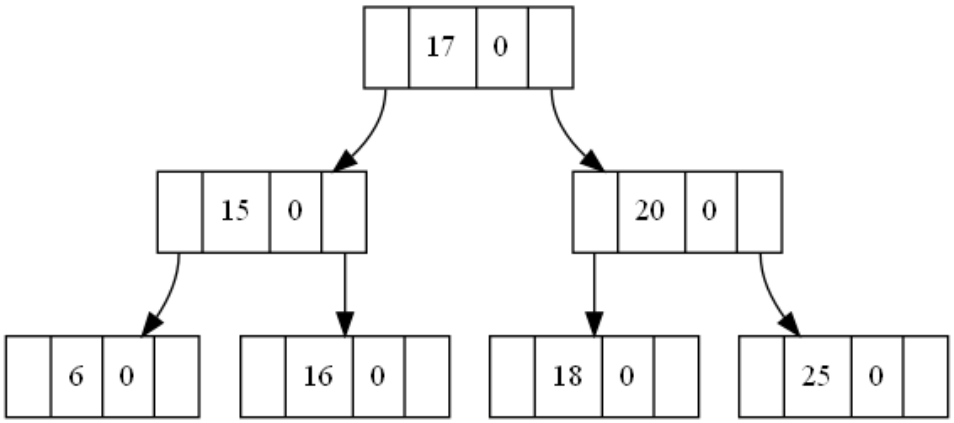
D2



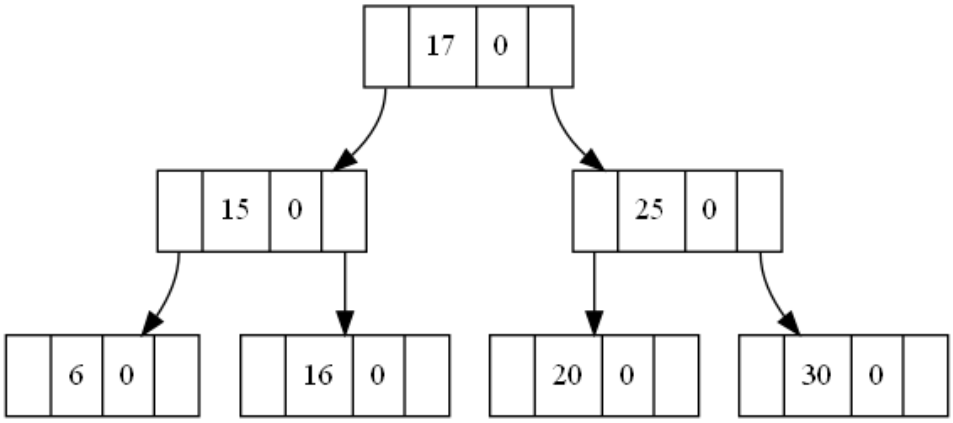
D3 – Deleting 9 from D2 makes a LL imbalance:



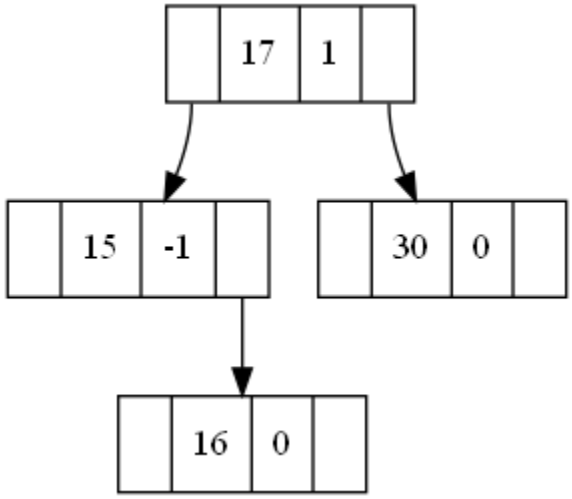
D4 – Deleting 7 ,8 from D3 creates a RL at root node.



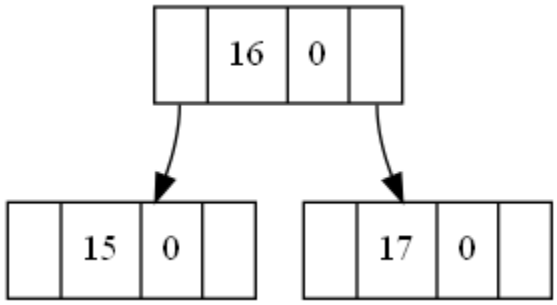
D5 - Inserting 30 and deleting 18 in d4 creates RR imbalance at node 20:



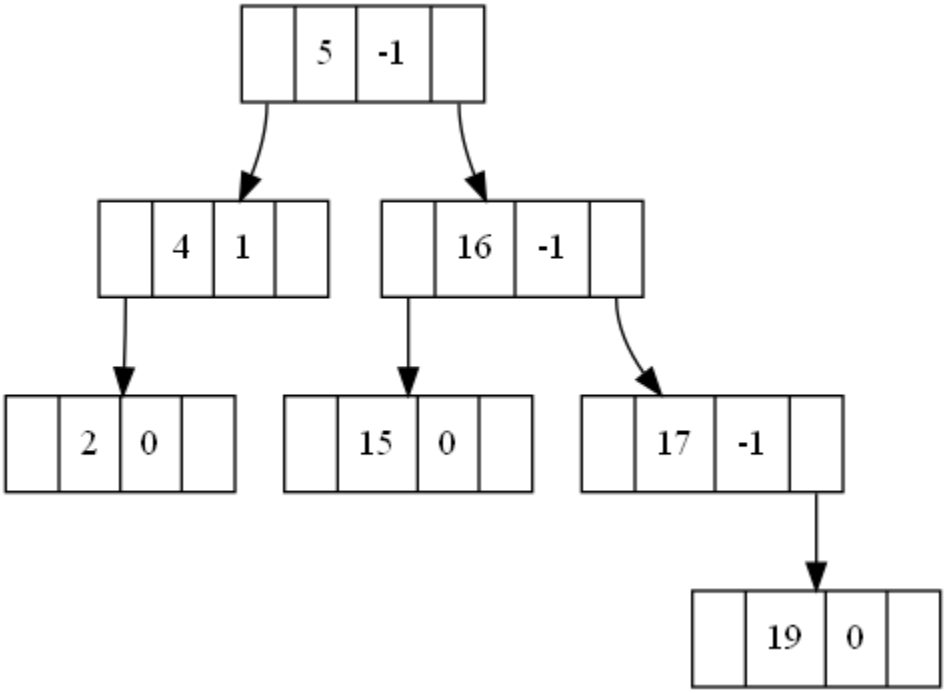
D6 – Deleting some nodes :



D7 : Deleting 30 creates LR imbalance at node 17 :

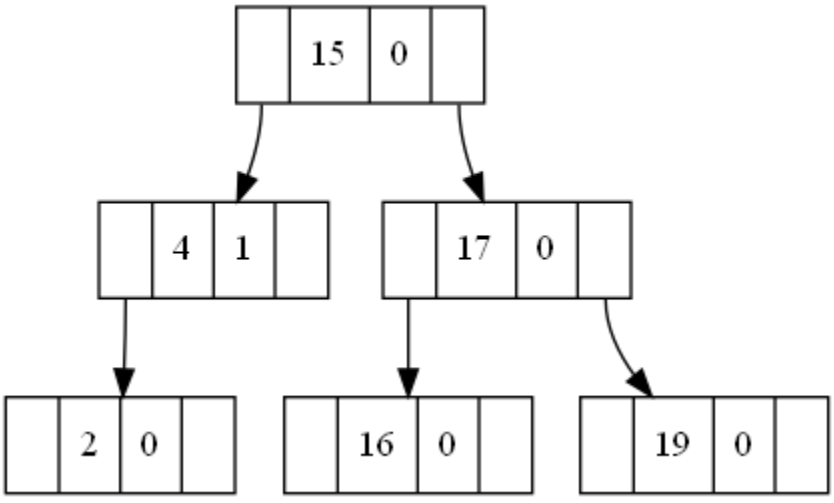


D8 – Inserting few Nodes



D9 – Deleting root node 5, root key replaced with key of node 15 and 15 is deleted after checking its children and adjusting if it has any.

This creates a RR Imbalance at root node 15 and is balanced accordingly.



End of Assignment . Thank You !!